

1 Einleitung

Um die Welt zu ruinieren, genügt es,
wenn jeder seine Pflicht tut.

Winston Churchill

Worum geht es in diesem Kapitel?

- Sie erhalten einen sehr komprimierten Überblick über das APM-Verfahren.
- Sie erfahren etwas über die Entstehung, Motivation und die Werte agiler Softwareentwicklung.
- Die organisatorischen Besonderheiten verschiedener Projektgrößen werden skizziert.
- Sie erhalten Hinweise zur Einführung und Adaption des APM-Verfahrens.

1.1 Das APM-Verfahren im Überblick

Zu Beginn des Projektes liegt das benötigte Ergebnis in einer Wolke. Es ist unscharf. Alle Beteiligten, Fachabteilung wie Entwickler, wissen nur grob, wie das Ergebnis aussehen soll. Es werden verschiedene Annahmen getroffen, die wahrscheinlich nur in Teilen zutreffend sein werden. Auf Basis dieser Annahmen wird das Projekt geplant, es wird ein Ziel angepeilt und dieses dann verfolgt.

Abnehmende Unschärfe

Anstatt nun die Augen zu verdrehen und darüber zu schimpfen, dass das Ziel in der Regel eher milchtrübe als glasklar ist, wird beim iterativen Vorgehen akzeptiert, dass die Klarheit über das herzustellende Produkt nicht mit einem Mal plötzlich entsteht, sondern schrittweise zustande kommt, und dass das Ziel keine konstante Größe ist, sondern sich mit der Zeit verändern kann.

Deswegen wird die Projektlaufzeit beim iterativen Vorgehen in eine Sequenz von Zeitfenstern eingeteilt, die man Iterationen nennt. Am Ende jeder Iteration wird innegehalten und zurückgeblickt:

- Was haben wir tatsächlich erreicht?
- Was wollten wir ursprünglich erreichen?
- Was lernen wir daraus?

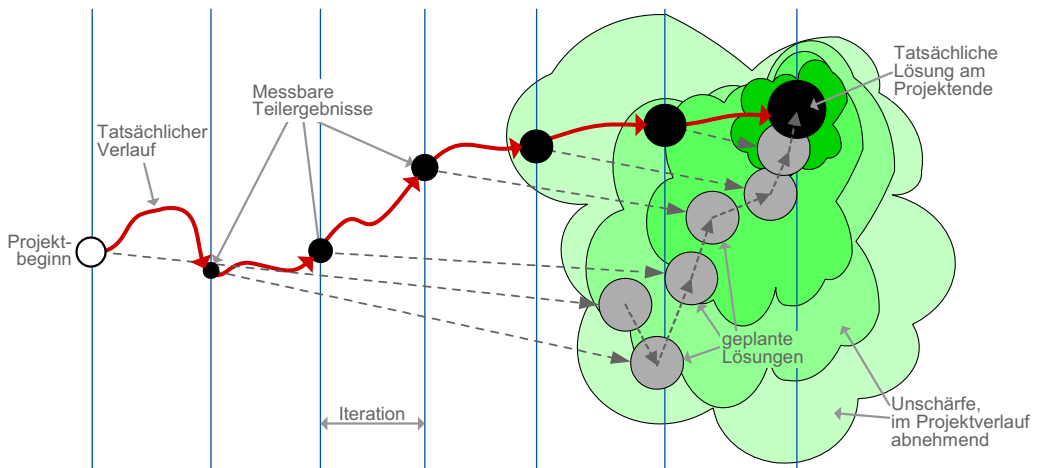


Abb. 1.1-1: Die Iterations-Wolken-Metapher: Schrittweise Zielklärung und -näherung (farbige Abbildung im PPT- und PDF-Format herunterladen unter www.oose.de/apm/download)

Dann wird der Blick wieder nach vorne gerichtet. Durch die zwischenzeitlich gewonnenen Erkenntnisse ist die Wolke etwas kleiner geworden. Das Ziel ist zwar immer noch unscharf, aber etwas weniger. Ausgehend von der in der abgeschlossenen Iteration tatsächlich erreichten Position wird nun ein neuer Plan gemacht und wieder das (nun etwas konkretere) Ziel angepeilt. Der Prozess beginnt von vorne.

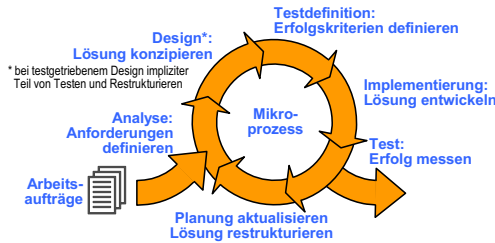


Abb. 1.1-2: Mikroprozessmodell einer Iteration

Mikroprozess
⇒ 178

Wichtig hierbei ist, dass in jeder Iteration prinzipiell alle elementaren Entwicklungsaktivitäten (Anforderungen definieren, Lösung konzipieren, Erfolgskriterien definieren, Lösung entwickeln, Erfolg messen und schließlich Planung aktualisieren) durchlaufen werden (siehe Abb. 1.1-2). Spätestens am Ende einer jeden Iteration steht also ein objektiv messbares Teilergebnis, ein Inkrement, d.h. eine teilfertige, vorübergehende, aber ausführbare Version der angestrebten Lösung.

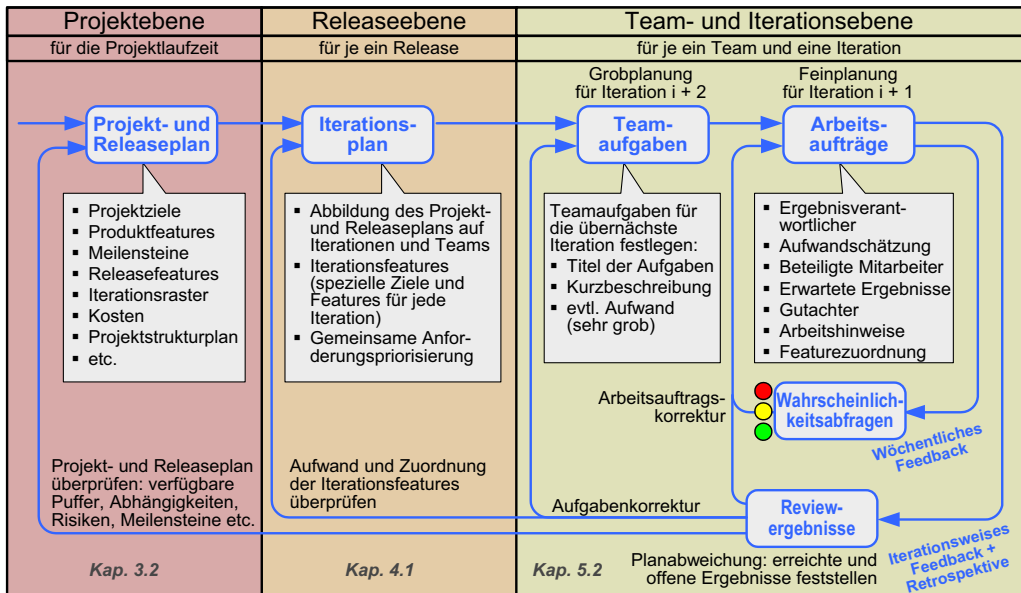


Abb. 1.1-3: Planungsebenen und Feedback-Schleifen (farbige Abbildung im PPT- und PDF-Format herunterladen unter www.oose.de/apm/download)

Wir verstehen Iterationen immer als sogenannte Timeboxen, d.h. einmal gestartet, wird der Endtermin einer Iteration nicht mehr verschoben, gerade auch dann nicht, wenn die Ergebnisse der Iteration hinter dem Plan zurückbleiben.

Iteration = Timebox
⇒190

Abb. 1.1-3 stellt diesen Regelkreis dar und zeigt die verwendeten Planungselemente und -ebenen im Überblick. Grundsätzlich unterscheiden wir dabei die Projektebene, die Releaseebene sowie die Team- und Iterationsebene.

- Die **Projektebene** enthält all jene Ergebnisse und Dokumente, die sich stets auf den gesamten Projektumfang beziehen. Sie werden während der gesamten Projektlaufzeit regelmäßig aktualisiert.

Projektebene ⇒122

Hierzu gehören die Projektziele, die Liste der herzustellenden Produktfeatures (also eine grobe Leistungsbeschreibung des oder der herzustellenden Produkte), Aussagen zu Kosten und geplanten Releases mit deren Features, Terminen und Meilensteinen sowie ein Iterationsraster (Anzahl und Dauer der Iterationen). Das wichtigste Ergebnis der Projektebene ist der **Projekt- und Releaseplan**.

- Die **Releaseebene** verfeinert die Ergebnisse der Projektebene dahingehend, dass für jedes Release die herzustellenden Releasefeatures auf die Iterationen und Teams aufgeteilt werden. Aus Releasefeatures werden **Iterationsfeatures** abgeleitet, um festzulegen, welche Ziele und Anforderungen in welcher Iteration von welchem Team zu bearbeiten sind, sodass das Release entsteht. Dabei werden die Anforderungen priorisiert, um deren Umsetzungsreihenfolge grob festzulegen. Das Ergebnis nennen wir **Iterationsplan**. Iterationsfeatures sind Zielvorgaben für die Teams.

Releaseebene ⇒135

- Die **Team- und Iterationsebene** verfeinert wiederum die Ergebnisse der Releaseebene. Für jedes Team und jede Iteration existieren grobe Zielbeschreibungen in Form von Iterationsfeatures. Nun gilt es, hieraus Aufgaben für einzelne Mitarbeiter und Teams abzuleiten. Dazu später mehr im Zusammenhang mit Abb. 1.1-6.

Team- und Iterations-
ebene ⇒161

Anhand der Abb. 1.1-3 wird auch deutlich, wie Erkenntnisgewinn in die Planung zurückgekoppelt wird (Feedback).

Der prinzipielle Weg der Verfeinerung von Features zum Arbeitsauftrag ist in Abb. 1.1-4 dargestellt. Die Aufteilung von Releasefeatures in Iterationsfeatures, die dann vollständig von einem Team und in genau einer Iteration entwickelt werden, ist dabei ein Idealfall. Die Abbildung zeigt, dass Releasefeatures in Iterationsfeatures für verschiedene Teams aufgeteilt werden können und dass gegebenenfalls bereits in einer vorigen Iteration Arbeitsaufträge als Zulieferleistung für ein Iterationsfeature existieren können. Ebenso ist zwar stets ein Team verantwortlich

für ein Iterationsfeature, dennoch können Zulieferleistungen aus anderen Teams oder gar anderen Teilprojekten erfolgen.

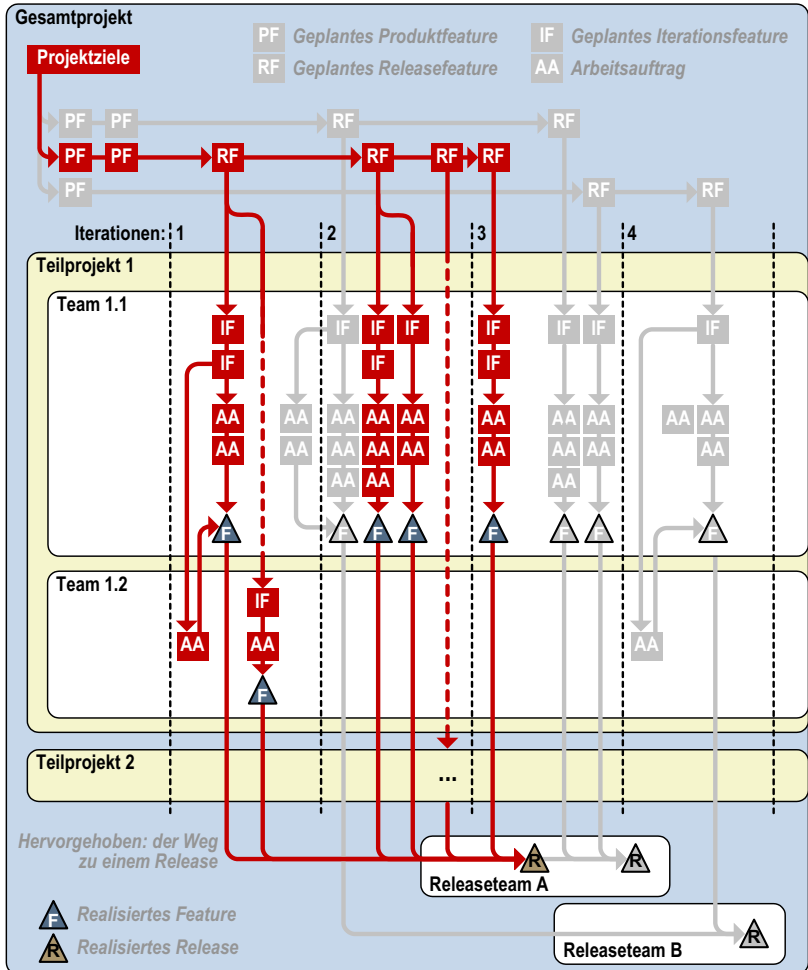


Abb. 1.1-4: Der Weg vom Projektziel über Produktfeatures, Releasefeatures und Iterationsfeatures zum Arbeitsauftrag

Produktfeatures
⇒ 138, 143, 154

Vorbereitungsphase ⇒ 63

Auf einer grobgranularen Ebene wird also das Gesamtprojekt in Form eines Projekt- und Releaseplans (vgl. Abb. 1.1-5) geplant, der im weiteren Projektverlauf kontinuierlich weiterentwickelt wird. Es werden Gesamtkosten, Produktfeatures, Endtermin, Releasetermine und Releasefeatures beschrieben.

Der Zeitraum, in dem die *erste Version* des Projekt- und Releaseplans entsteht, nennen wir **Vorbereitungsphase**, weil wir diese Informationen gewöhnlich benötigen, bevor wir ein Angebot abgeben können oder einen Auftrag bekommen.

Iterationen	1	2	3	4	5	6	7	8	9	10	11	12	13	14		
Starttermin	28.01.	10.03.	21.04.	02.06.	14.07.	25.08.	06.10.	20.10.	01.12.	12.01.	23.02.	06.04.	18.05.	29.06.	13.07.	24.08.
Stabilisierungsiteration							1							2		
Phasen																
Startphase (M1)	bis 20.04.		▲													
Hauptphase (M2)												bis 12.07.		▲		
Abschlussphase (M3)															bis 02.10.	
Meilensteine																
Technischer Prototyp (M4)	▲															
Vorabnahmen																
Benutzungskonzept (M5)	▲															
Fahrzeugreservierung (M6)				am 14.07.	▲											
Fahrtdatenverarbeitung (M7)				am 14.07.	▲											
Rechnungserstellung (M8)			am 02.06.	▲												
Fahrzeugdisposition (M9)					am 25.08.	▲										
Stationsverwaltung (M10)										am 12.02.	▲					
Fuhrparkverwaltung (M11)								am 01.12.	▲							
GPS-Tracking (M12)								am 01.12.	▲							
Kundenverw. komplett (M13)											am 01.05.	▲				
Tarif neu: Berechnung (M14)									am 22.12.	▲						
Tarif neu: Stammdaten (M15)											am 01.05.	▲				
Internetreservierungen (M16)											am 06.04.	▲				
Releases																
Verwaltung Basis (M17)					am 06.10.	▲										
Internetreservierungen (M18)												am 18.05.	▲			
Verwaltung komplett (M19)												am 29.06.	▲			
Verwaltung korrigiert (M20)														am 24.08.	▲	
Abnahmen																
Verwaltung Basis (M21)								am 01.12.	▲							
Verwaltung komplett (M22)															am 21.09.	▲
Internetsystem (M23)												am 20.06.	▲			

Abb. 1.1-5: Projekt- und Releaseplan mit Iterationsraster, Phasen und Meilensteinen für Vorabnahmen, Releases und Abnahmen

Die eigentliche Projektlaufzeit wiederum teilen wir nun in ein Raster etwa gleich langer **Iterationen** auf (im Sinne von **Timeboxen**). Dazu benötigen wir den Endtermin und die Entscheidung darüber, wie lange eine Iteration sein soll. Über dieses Iterationsraster werden zur Orientierung noch folgende drei grundsätzliche zeitliche Abschnitte gelegt (vgl. Abb. 1.1-5):

- Die **Startphase**, in der noch viele grundsätzliche Fragen und Entscheidungen ungeklärt oder instabil sind (besteht aus einigen wenigen Iterationen). Die Planung wird spätestens jetzt und mindestens für das erste Release bis auf Releaseebene herunter geplant. Startphase ⇒ 113
- Die **Hauptphase**, in der auf Basis von als stabil zu betrachtenden Entscheidungen und Strukturen die eigentliche Entwicklung erfolgt (erstreckt sich über den Hauptteil der Iterationen) und bereits ver- Hauptphase ⇒ 118

schiedene Releases entstehen. Das heißt, schrittweise werden bereits Systemteile eingeführt und gegebenenfalls auch vom Kunden abgenommen.

Abschlussphase ⇒ 119

- Die **Abschlussphase**, in der das Produkt (soweit noch nicht in der Hauptphase geschehen) eingeführt und dem Auftraggeber übergeben wird (erstreckt sich über eine oder wenige Iterationen).

Was in Abb. 1.1-3 als Team- und Iterationsebene aus planungstechnischer Sicht dargestellt wird, spielt sich größtenteils innerhalb einer Iteration ab, deren Ablauf wir nun näher spezifizieren. Abb. 1.1-6 gibt einen grafischen Überblick und zeigt den grundsätzlichen Aufbau einer Iteration. Dabei wird die Iteration zunächst in einen (längeren) Fortschrittsabschnitt und einen (kürzeren) Orientierungsabschnitt unterteilt.

Iterationsfeatures
⇒ 146, 151, 155

- Ausgehend vom **Iterationsplan**, der Iterationsziele und **Iterationsfeatures** enthält, wird nun die Planung iterationsweise verfeinert:
 - ♦ Um diese Detailplanung schon etwas vorzustrukturieren, findet für die jeweils übernächste **Iteration (i+2)**, also vorher, bereits eine Grobplanung statt. Hier werden auf Basis von Iterationsfeatures team- und iterationspezifisch die Arbeitsaufträge in Form abstrakter **Teamaufgaben** identifiziert, d.h. lediglich mit Titel, Kurzbeschreibung und gegebenenfalls mit einer ganz groben Aufwandschätzung versehen.
 - ♦ Für die jeweils nächste **Iteration (i+1)** legt jedes Team feingranular die **Arbeitsaufträge** fest, die innerhalb der Iteration erledigt werden sollen. Ein Arbeitsauftrag beschreibt, wer für das Ergebnis verantwortlich ist, welche weiteren Personen daran beteiligt sind, wie das Ergebnis aussehen und abschließend beurteilt werden soll, wer das Ergebnis abschließend begutachten soll und wie viel Aufwand insgesamt für den Arbeitsauftrag geschätzt wird.

Zwei-Bugwellen-Planung
⇒ 162, 166

Diese teamspezifische Grobplanung (Iteration i+2) und Feinplanung (Iteration i+1) nennen wir die **Zwei-Bugwellen-Planung**, weil sie wie Bugwellen vor der aktuellen Iteration entstehen und iterationsweise aktualisiert werden.

Tägliches Teamsteuerungstreffen ⇒ 182, Stehung
⇒ 345

- Während der Iteration koordiniert und synchronisiert jedes Team selbstverantwortlich die eigene Arbeit durch kurze **tägliche Teamsteuerungstreffen** (Daily-Scrum-Meetings, Stehungen).

Tägliches Build ⇒ 181

- Möglichst kontinuierlich oder zumindest täglich ist eine unvollständige, aber prinzipiell lauffähige Version der Software zu bauen und zu testen (**tägliche (Smoke-)Builds**). Je nach vertretbarem Aufwand und Machbarkeit sind alle Neuerungen und Änderungen zumindest teamweit, möglichst aber auch projektweit zu integrieren. So entstehen in sehr kurzen Zyklen **Alpha-Versionen** der Software (Always Alpha).

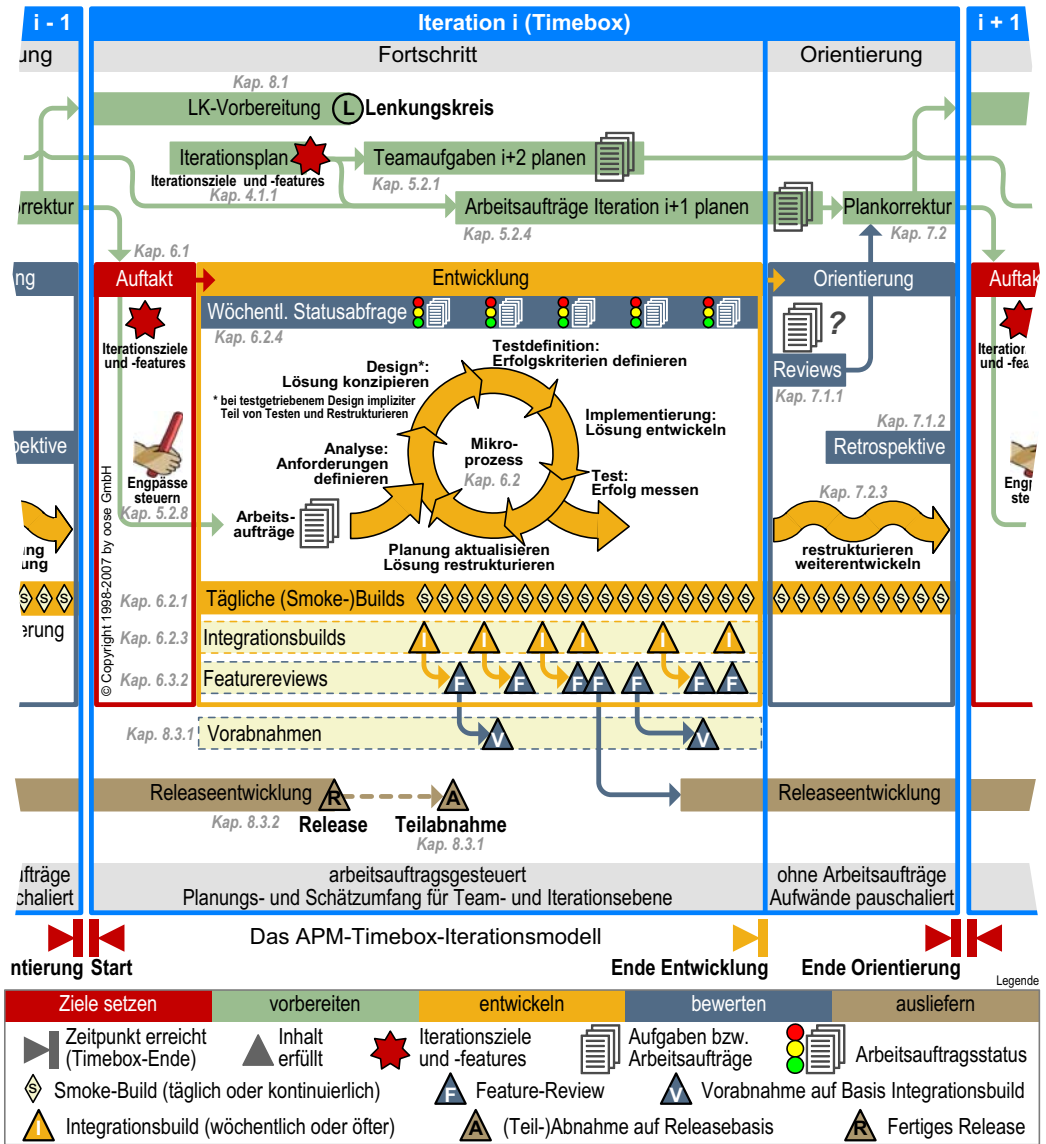


Abb. 1.1-6: Prinzipieller Aufbau einer Iteration (farbige Abbildung im PPT- und PDF-Format herunterladen unter www.oose.de/apm/download)

- Einmal wöchentlich werden systematisch teamübergreifend Problemsituationen gesucht, um innerhalb einer Iteration den Plan realistisch zu halten und auch übergeordnet intervenieren zu können (wöchentliche arbeitsauftragspezifische Statusabfragen, **Ampelstatus**, Wahrscheinlichkeitsabfragen, Läufer).

Ampelstatus ⇒ 185, 347
 Läufer ⇒ 349

- Arbeitsauftragsreviews
⇒198
Planungskorrektur ⇒202
- Am Ende einer jeden Iteration führen alle Teams zeitgleich eine Bestandsaufnahme (Soll-Ist-Abgleich) für alle bearbeiteten Arbeitsaufträge durch (**Arbeitsauftragsreviews**). Mit diesen Erkenntnissen wird die bereits vorliegende Feinplanung (teamspezifische Arbeitsaufträge) für die unmittelbar folgende Iteration abgeglichen und angepasst (**Planungskorrektur**).
- Retrospektive ⇒201
- Ebenfalls am Ende der Iteration führen alle Teams und sonstigen Organisationseinheiten des Gesamtprojektes jeweils eigene **Retrospektiven** durch, mit denen geklärt werden soll, was gut lief und wie die zukünftige Arbeit verbessert werden kann.
- Releaseentwicklung ⇒221
- Um ein **Release** herzustellen, wird die erste potenzielle Alpha-Version, die alle geforderten Releasefeatures erfolgreich und stabil umgesetzt hat, aus dem Entwicklungsprozess herausgenommen und zu einem Release weiterentwickelt. Gegebenenfalls können Releases auch von separaten eigenständigen **Releaseteams** entwickelt werden. Bei Bedarf können auch mehrere Releases gleichzeitig bzw. zeitlich überlappend entstehen (**Parallelreleases**).
- Teilabnahmen ⇒220
- Releases können zu vertraglich relevanten **Teilabnahmen** führen.
- Vorabnahmen ⇒220
- Unabhängig von Releases können auf der Basis von Alpha-Versionen (vorbehaltlich der Reproduzierbarkeit in einer Releaseversion) vertraglich relevante **Vorabnahmen** stattfinden.

So weit in aller Kürze der Überblick. Wie dieses Verfahren nun im Detail funktioniert, welche Varianten und andere wichtige Aspekte es gibt, das erfahren Sie in den übrigen Kapiteln dieses Buches.

Sie werden dort einen relativ festen und klaren Rahmen, Anleitungen und Hinweise zu speziellen Vorgehensweisen finden sowie viele einzelne Handlungsanweisungen, die miteinander verzahnt sind und aufeinander aufbauen: Tue dies und tue das und dann das. Manchmal erscheint Ihnen dies möglicherweise sogar bürokratisch.

Bitte beachten Sie jedoch die in Kapitel 1.3 *Werte* (⇒20) beschriebenen Werte und die in den Kapiteln 1.4 *Projektorganisation und Projektgrößen* (⇒30) und 1.5 *Einführung und Adaption des Verfahrens* (⇒41) genannten Grundlagen und Rahmenbedingungen. Ziel dieser Rahmenbedingungen und Vorgehensweise ist es, ein Umfeld und Rahmenbedingungen zur guten Entfaltung agiler und eigenverantwortlicher Projektarbeit zu schaffen. Wir definieren in diesem Kapitel bewährte Spielregeln, grenzen Aufgaben und Verantwortlichkeiten ab, damit sich innerhalb dieses Rahmens ohne unnötige Konflikte und mit klaren Freiräumen und Verantwortlichkeiten effizient und effektiv arbeiten lässt.

Die Ausgangssituationen, Projektkulturen, Werte, vorhandenen Erfahrungsschätze etc. sind in großen Projekten sehr unterschiedlich. Was in einem Umfeld selbstverständlich ist, kann in einem anderen Umfeld revolutionär oder fragwürdig sein. Und wenn wir hier eine Vorgehensweise beschreiben, die sich vielfach bewährt hat, kann sie genau in Ihrem Projekt möglicherweise völlig unpassend sein. Also verstehen Sie den in diesem Buch beschriebenen Prozess als Anregung und Ausgangsbasis – aber es bleibt Ihre eigene Verantwortung zu entscheiden, ob oder wie dieser Prozess in Ihrer Organisation wirklich gut funktionieren kann.

Wir betrachten den hier beschriebenen Prozess als vielfach bewährt und praxistauglich, und doch unterscheiden sich alle uns bekannten Praxisbeispiele in ihrer ganz konkreten Ausgestaltung. Deswegen geben wir immer wieder Hinweise auf Alternativen und typische Hindernisse oder auf Abhängigkeiten von bestimmten Faktoren.

Die wichtigsten Faktoren hierbei sind:

■ Projektgröße

Vgl. Fallbeispiel ⇒65

Um das Verständnis zu erleichtern, fokussieren wir in der Darstellung unseres Verfahrens vorwiegend auf ein Großprojekt mit einer Iterationsdauer von sechs Wochen, einer Gesamtdauer von 20 – 24 Monaten und ca. vier Releases bis zum Endprodukt. Unser Fallbeispiel basiert auf ca. 25 Projektmitgliedern. Darauf aufbauend beschreiben wir dann bedarfsweise, welche Änderungen oder Einschränkungen zu beachten sind, wenn andere Rahmenbedingungen vorliegen.

■ Auftraggeber-Auftragnehmer-Situation

Eine unternehmensinterne Entwicklung bringt andere Herausforderungen mit sich als beispielsweise ein harter externer Auftraggeber. Je nach Situation sind andere Vorgehensweisen sinnvoll oder wichtig. In unserem Fallbeispiel beziehen wir uns auf einen externen Auftraggeber.

■ Soziale Beziehungen

Haben Auftragnehmer und Auftraggeber aus vergangenen Vorhaben ein belastbares, krisenfestes vertrauensvolles Verhältnis? Kennen sich die Beteiligten schon persönlich? Ist das Projektteam eingespielt oder ist es ein bunter Haufen Unbekannter? Werden Konflikte unter den Teppich gekehrt statt konstruktiv ausgetragen?

■ Fachliche Sicherheit

Wie gut kennen die Projektmitarbeiter die Fachlichkeit, die Ziele und das Umfeld des Kunden?

■ Werte und Nachhaltigkeit

Gerade in Großprojekten findet sich keine heile Welt, und die herrschenden Werte widersprechen oftmals jeglichen sozialromantischen Idealen. Egal, ob man solche Ideale vertritt oder im Gegenteil sogar hemmungslos mitspielt, die Werte und Spielregeln sollten von der Projektleitung zumindest erkannt und verstanden und möglichst kompetent abgefangen werden. Aus diesem Grund erwähnen wir in der Werkzeugsammlung in Kapitel 9 beispielsweise auch einige zweiseitige Managementstrategien.

Für die Einführung des APM-Verfahrens werden keine speziellen Werte vorausgesetzt, es forciert und fördert jedoch bestimmte Werte. Je nach Ausgangssituation gestaltet sich die Einführung des Verfahrens etwas anders.

1.2 Agile Softwareentwicklung

1.2.1 Historische Missverständnisse

Winston Royce

Iterative Projektmanagementverfahren sind so alt wie die Informatik. Bereits 1970 publizierte Winston Royce ein erstes Modell [Royce-1970]. Diese Publikation wurde wiederum verwendet bei der Entwicklung des US-Militär-Standards DoD-STD-2167.

David Maibor

Der 2167-Mitautor David Maibor, der sich auf die Arbeit von Winston Royce bezog, war jedoch mit iterativ-inkrementeller Entwicklung und evolutionären Anforderungen nicht vertraut. Stattdessen berief sich Maibor auf eine spezielle und stark vereinfachte Variante des Modells von Royce, die mit 1 – 2 Iterationen auskam. So legte Maibor mit dieser Vereinfachung die Grundlagen des Wasserfallmodells.

STD-2167 wurde wiederum Grundlage für andere Vorgehensmodelle, wie etwa CMMI (Capability Maturity Model Integration) oder das deutsche V-Modell.

Wasserfallmodell ist ein historischer Irrtum

Walker Royce, der Sohn von Winston Royce, berichtete später, sein Vater wäre immer ein Vertreter von iterativen, inkrementellen und evolutionären Entwicklungsmodellen gewesen. Sein Arbeitspapier hätte das Wasserfallmodell lediglich als die einfachste Möglichkeit, die aber nur für die unkompliziertesten Projekte funktioniert, beschrieben. Und auch Maibor hat später geäußert, dass er, wäre er damit damals vertrauter gewesen, das iterative Vorgehen im STD-2167 viel deutlicher emp-

fohlen hätte [Larman-2004b]. So gesehen handelt es sich beim Wasserfallmodell um einen historischen Irrtum, der bekanntermaßen problematische Effekte erzeugt, insbesondere wenn man das Wasserfallmodell unreflektiert auf große Projekte anwendet.

Interessant ist dabei, dass Winston Royce den Begriff „Wasserfall“ überhaupt nicht benutzt. Wer seine Veröffentlichung jedoch liest, dem wird unmittelbar klar, warum es sich um ein „Wasserfallmodell“ handelt. Das Papier von Winston Royce ist übersät mit Abbildungen, in denen kaskadenartige (=wasserfallartige) Phasen visualisiert werden. Mehr zu den Grundlagen des Wasserfallmodells findet sich in [Himmelreich-2006].

Warum der Erfinder des Wasserfallmodells missverstanden wurde

In den 1980er- bis 1990er-Jahren verbreitete sich das Wasserfallmodell innerhalb der Informatik sehr rasant und wurde Praxisstandard. Das iterative Modell wurde weiterentwickelt, hatte es jedoch schwer, sich durchzusetzen, obwohl es beachtliche Erfolge damit gab. So entwickelte beispielsweise die IBM von 1977 – 1980 die Software für das NASA Space-Shuttle in 17 Iterationen über 31 Monate mit durchschnittlich achtwöchigen Iterationen [Madden-1984].

Space-Shuttle-Software iterativ entwickelt

Etwas später, 1985, publizierte Barry Boehm das sogenannte Spiralmodell [Boehm-1985]. Dieses Modell, das ebenfalls einen iterativ-inkrementellen Ansatz verfolgt, wurde theoretisch viel beachtet und anerkannt, in der Praxis jedoch selten angewendet. Der irrtümliche Standard Wasserfallmodell galt als bodenständiger.

Spiralmodell

Ende der 1990er-Jahre erblühten dann verschiedene neue Varianten des iterativen Vorgehens. Am bekanntesten wurde Extreme Programming (XP, [Beck-2003]). Im Jahre 2005 schließlich wurde die erste Version des V-Modell XT veröffentlicht, das offizielle Vorgehensmodell in der Bundesrepublik Deutschland, in dem erstmalig auch iterativ-inkrementelle und agile Projektdurchführungsstrategien enthalten sind. Spätestens seit diesem Zeitpunkt sind agile Managementtechniken auch im Rahmen öffentlicher Aufträge anwendbar.

Das iterative Vorgehen wurde Anfang der 2000er-Jahre immer populärer in der Praxis. Immer mehr Studien belegten schließlich auch die Vorteile und Erfolge dieses Ansatzes, beispielsweise der regelmäßige sogenannte Chaos-Report der Standish Group [Standish-Chaos]. Dazu mehr in Kapitel 1.5.3 *Faktoren für erfolgreiche Projekte* (⇒47).

Chaos-Report

1.2.2 Entstehung und Motivation

Hinter den Schlagworten *agile Softwareentwicklung* oder *agiles Projektmanagement* verbirgt sich eine Ende der 1990er-Jahre entstandene Gegenbewegung zu den überreglementierten und starren Ansätzen der 1980er- und 1990er-Jahre.

Standardisierung

Seit Anbeginn der Informatik existieren Bestrebungen, Softwareentwicklung erfolgreicher zu machen. Ein Mittel hierfür ist die Standardisierung von Prozessen beispielsweise mithilfe von Vorgehensmodellen. Die Grundidee dabei ist, bewährte Techniken und Vorgehensweisen als Mindeststandards konkret festzuschreiben und zu standardisieren. Bekannte Fehler und Problemsituationen sollen damit vermieden werden, was vor allem für Unerfahrene nützlich ist.

Schattenseiten

Grundsätzlich funktioniert dieser Verbesserungsansatz, aber er hat auch Schattenseiten:

- Sehr erfahrene und erfolgreiche Projektmanager werden gelegentlich behindert. Es erfolgt im Zweifelsfall eine Standardisierung auf Mittelmaß.
- Projekte mit üblichen, durchschnittlichen und von den Vorgehensmodellen vorgesehenen Problemsituationen erhalten eine gute Unterstützung. Für alle anderen Projekte kann der Standard unpassend, möglicherweise sogar kontraproduktiv sein.

Projektbürokratie

- Vorgehensstandards können zum bürokratischen Selbstzweck verkommen, wenn die dafür Verantwortlichen den Praxisbezug verloren haben.
- Standards werden oftmals zu langsam weiterentwickelt und zielen somit auf Problemsituationen, die in der Vergangenheit eine Relevanz hatten, unterstützen aber die aktuellen Fragestellungen unzureichend.
- Die beteiligten Personen übernehmen weniger eigene Verantwortung für ihre Entscheidungen und ihr Handeln, da sie schließlich nur Vorschriften befolgen.

Die sogenannte agile Entwicklung ist eine Gegenbewegung hierzu, die die Vorteile und Errungenschaften einerseits erhalten will, die aber andererseits die erkannten Begrenzungen auflösen und darüber hinausgehen möchte.

Manchmal schließen sich solchen Bewegungen die falschen Personen an. Die agile Entwicklung wurde initiiert von sehr erfahrenen Personen, die an einer Weiterentwicklung und Innovation interessiert sind. Sie sollten nicht verwechselt werden mit den Protagonisten, die aus Be-

quemlichkeit die mühevollere Weiterentwicklung und das Lernen scheuen und ihre Unerfahrenheit oder Stagnation damit rechtfertigen, Agilität als Beliebigkeit zu interpretieren.

Der Wortlaut in den folgenden Abschnitten ist durchaus ernst zu nehmen. Wenn es heißt "Höchste Priorität hat die Zufriedenstellung des Auftraggebers durch frühe und kontinuierliche Lieferung brauchbarer Software", dann ist "brauchbar" nicht zu interpretieren als perfekt oder optimal, sondern einfach nur als brauchbar.

Und wenn es heißt, "Funktionierende Software ist wichtiger als umfangreiche Dokumentation", dann bedeutet dies nicht, dass Dokumentation unwichtig oder zu vermeiden ist, sondern dass die beste und umfangreichste Dokumentation keinen Wert hat, wenn die Software nicht funktioniert.

Beim Thema Standardisierung ist außerdem zu unterscheiden, ob Standards abstrakt von außen vorgegeben werden und deswegen möglicherweise für die projektspezifische Situation nicht passen oder ob es Vereinbarungen und Standards sind, die sich das Projekt selbst gegeben hat. Im letzteren Fall ist die Disziplin der beteiligten Projektmitarbeitenden zu erwarten.

1.2.3 Das agile Manifest

Das agile Manifest (siehe <http://www.agilemanifesto.org/>) wurde im Jahr 2001 von Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland und Dave Thomas im Rahmen eines Treffens in Snowbird, Utah, ins Leben gerufen und innerhalb weniger Wochen von Hunderten anderer bekannter Persönlichkeiten der Branche unterzeichnet.

Der Wortlaut des Manifestes:

„Wir entdecken bessere Wege zur Entwicklung von Software, indem wir Software entwickeln und anderen bei der Entwicklung helfen. Durch diese Tätigkeiten haben wir gelernt:

- Individuen und Interaktion sind wichtiger als Prozesse und Werkzeuge
- Funktionierende Software ist wichtiger als umfangreiche Dokumentation
- Kooperation mit Projektbetroffenen ist wichtiger als Vertragsverhandlungen
- Reaktion auf Änderungen ist wichtiger als Festhalten an einem starren Plan

Natürlich sind auch die Dinge rechts wichtig, aber im Zweifelsfall schätzen wir die linken höher ein.“

Hier ein paar Erläuterungen zum agilen Manifest:

■ **Individuen und Interaktion sind wichtiger als Prozesse und Werkzeuge**

Die Menschen stehen im Mittelpunkt agiler Methoden und gelten als wichtigster Erfolgsfaktor in Projekten. Auf die Motivation und gute Zusammenarbeit im Team wird Wert gelegt. Die Eigenverantwortung der Entwickler und der übrigen Projektbeteiligten wird betont und aktiv gefördert. Das macht die Leichtgewichtigkeit überhaupt möglich. Auf starr reglementierte Prozesse, welche die Menschen aus der Verantwortung nehmen, kann verzichtet werden. Es genügt, nur das Nötigste zu planen und vorzuschreiben.

■ **Funktionierende Software ist wichtiger als umfangreiche Dokumentation**

Laufende Software ist wichtiger als Dokumentation. Nur die wirklich notwendige Dokumentation wird erstellt. Eine gute Kommunikation zwischen den Projektbeteiligten reduziert die Notwendigkeit für umfangreiche Dokumentation. Funktionierende Software wird in kurzen Abständen von ca. 1 bis 3 Monaten an den Kunden ausgeliefert oder diesem zumindest zur Evaluierung bereitgestellt. Ein erstes Release sollte kurz nach Projektstart ausgeliefert werden. Das Projekt erhält somit direktes Feedback, Änderungen sind schnell möglich und reduzieren somit den Aufwand an Anforderungsdokumentation. Zusätzlich übt das Team den Auslieferungsprozess, dieser wird Normalität.

Sofern als Projektergebnis auch eine Dokumentation beispielsweise für die Wartung und Weiterentwicklung gewünscht ist (Dauerdokumentation), wird diese selbstverständlich auch in agilen Prozessen erstellt.

■ **Kooperation mit Projektbetroffenen ist wichtiger als Vertragsverhandlungen**

Die Zusammenarbeit mit dem Kunden steht im Mittelpunkt agiler Ansätze. Eine enge Zusammenarbeit mit dem Kunden wird angestrebt, auch um den Kunden stärker in die Verantwortung zu nehmen. Das Vertrauensverhältnis und die enge Zusammenarbeit sind wichtig. Der Kunde soll maßgeblich mitentscheiden, welche Features in welchen Releases enthalten sein sollen. Durch die frühzeitige und kontinuierliche Lieferung bereits brauchbarer Software wird das Risiko, die falsche Software zu entwickeln, erheblich reduziert. Dadurch sinkt auch die Bedeutung detaillierter Verträge. Nicht gemeint ist, vollständig auf Verträge zu verzichten, sondern den Regelungsbedarf durch eine auf Vertrauen ausgelegte Zusammenarbeit zu minimieren.

■ **Reaktion auf Änderungen ist wichtiger als Festhalten an einem starren Plan**

Während der Projektlaufzeit ergeben sich typischerweise Änderungen und neue Anforderungen. Auf diese soll flexibel reagiert werden. Agile Methoden versuchen, flexibel mit Änderungswünschen umzugehen. Die Planung und Anforderungsdefinition erfolgen evolutionär, das heißt, sie werden schrittweise verfeinert. Trotzdem existiert eine Gesamtplanung, allerdings nur so detailliert, wie es für das Sicherheitsbedürfnis der Beteiligten notwendig ist.

1.2.4 Prinzipien agiler Softwareentwicklung

Folgende Prinzipien ergeben sich direkt aus dem Manifest bzw. wurden daraus abgeleitet:

- Höchste Priorität hat die Zufriedenstellung des Kunden durch frühe und kontinuierliche Lieferung brauchbarer Software.
- Anforderungsänderungen sind auch in fortgeschrittenen Entwicklungsstadien möglich.
- Die Software wird inkrementell und in kurzen Iterationen erstellt.
- Fachexperten und Entwickler arbeiten möglichst direkt und täglich zusammen.
- Die effizienteste und effektivste Art, Informationen zu verbreiten, ist die direkte Kommunikation von Angesicht zu Angesicht.
- Funktionierende Software ist die primäre und wichtigste Kenngröße für den Projektfortschritt.
- Konzentration auf das Wesentliche heißt explizit und regelmäßig zu entscheiden, was wegzulassen ist.
- Das Entwicklungsteam reflektiert in regelmäßigen Abständen darüber, wie es die gemeinsame Arbeit verbessern kann.

Wir möchten hier nicht detailliert die Grundprinzipien wiederholen, die in anderen guten Büchern zur agilen Softwareentwicklung bereits umfassend beschrieben sind, und verweisen für weitere Details auf das Buch „Agile Software-Entwicklung“ von Alistair Cockburn [Cockburn-2003].

„Agile Software-Entwicklung“ von Alistair Cockburn

1.2.5 Bezug zu anderen agilen Verfahren

Scrum, XP, Crystal

Es gibt derzeit kein agiles Standardverfahren, und wahrscheinlich wäre dies auch nicht zielführend. Bekannte Verfahren sind Extreme Programming (XP) [Beck-2003, Wolf-2005], Scrum [Schwaber-2004, Beedle-2002, Schwaber-2007], die Crystal-Methodiken [Cockburn-2003] und der Eclipse-Way [Gamma-2007]. Diese Verfahren sind (mit Ausnahme des Eclipse-Way) ebenso wie unser APM-Verfahren Mitte/Ende der 1990er-Jahre erstmalig publiziert worden. In der Praxis des deutschen Sprachraums gehören Scrum und XP zu den beliebtesten Verfahren.

Viele Elemente, Techniken und Werte dieser Verfahren sind ähnlich. Während zu Beginn der agilen Bewegung die Unterschiede beispielsweise bezüglich typischer Anwendungsgebiete, adressierter Projektgrößen oder Beschränkungen auf spezielle Teildisziplinen oder Projektrollen noch deutlich waren, haben sich in den letzten 5 – 7 Jahren alle Verfahren weiterentwickelt und sind umfassender, allgemeiner sowie praktisch und theoretisch fundierter geworden.

Das APM-Verfahren hat die größten Ähnlichkeiten wahrscheinlich mit Scrum, vor allem bei den Techniken auf der Mikroebene. Scrum macht in diesem Bereich klare Vorgaben beispielsweise bezüglich der Rollen, Dauer von Sprints und spezieller Meetings.

Es existieren viele Ähnlichkeiten

APM beinhaltet Elemente aus Scrum, die aufgrund der eigenständigen Entwicklung teilweise jedoch unter anderem Namen oder in anderer Ausprägung existieren. Scrum beinhaltet eine eigene Terminologie (bspw. *Product Backlog*, *Sprints*, *Sprint Backlog*), während wir im APM-Verfahren uns eher an traditioneller Terminologie orientieren (bspw. *Arbeitsauftrag*, *Featureliste*, *Releaseplan*).

Das APM-Verfahren lehnt sich also bezüglich Methodik, Konzepten und Terminologie so weit wie möglich an klassischen Projektmanagementverfahren an. Ausgangspunkt ist daher nicht die Unterstützung von Entwicklerteams durch Mikromanagementtechniken, sondern ein umfassendes Projektmanagement.

APM ist die agile Erweiterung bewährter Verfahren

APM verstehen wir als eine Erweiterung und Ergänzung traditioneller Projektmanagementansätze und bauen auf diesen auf.¹ Der international bekannteste Ansatz stammt vom Project Management Institute (PMI). Das PMI bietet eine PMP (Project Management Professional) genannte Zertifizierung an, die an eine relativ hohe Eintrittsschwelle

PMI/PMP

¹ Die 5-tägige APM-Ausbildung von oose ist vom PMI in der Weise anerkannt, dass zertifizierte PMPs durch die Teilnahme am APM-Training Punkte zum Erhalt ihres PMP-Status erwerben können.

geknüpft ist. Neben dem PMI hat auch die GPM (Gesellschaft für Projektmanagement) eine Relevanz und hohe Wertigkeit.

Agilität heißt Beweglichkeit. Beweglichkeit setzt Freiräume und Freiheiten voraus, das heißt Abweichung von Standards oder vorgegebenen vielleicht starren, aber möglicherweise auch bewährten Regelwerken. Wo immer Freiheit entsteht, gibt es Verantwortung als Kehrseite der Medaille. Freiheit ohne Verantwortung ist egoistisch und einseitige Interessenverfolgung. Verantwortung wiederum setzt Kompetenz voraus.

Verantwortung als Kehrseite von Freiheit und Agilität

Bezogen auf agiles Projektmanagement heißt dies für uns, dass wir von agilen Projektteams und Führungskräften in agilen Projekten eine höhere Projektmanagementkompetenz erwarten als von solchen aus herkömmlichen Projekten. APM sehen wir, zumindest für mittlere und größere Projekte, nicht als Einstieg in das Thema Projektmanagement, sondern als Zusatzqualifikation.

APM baut auf herkömmlichem PM auf

Erst auf dieser Grundlage, so unsere Erfahrung, gelingt es, auch große Projekte mit 50, 100 oder 200 Projektmitgliedern so aufzusetzen, zu planen, zu steuern und zu führen, dass sie durch die Nutzung agiler Verfahren erfolgreicher werden als ohne agile Techniken.

In der Praxis lässt sich das APM-Verfahren mit Vorgehensmodellen wie dem OEP (oose Engineering Process, [OEP-2007]) oder dem V-Modell XT kombinieren.

Die im APM-Verfahren optional verwendeten Features mit ihren speziellen Ausprägungen Releasefeature, Iterationsfeature etc. haben keinen direkten Bezug zu dem von Jeff DeLuca entwickelten *Feature Driven Development* (FDD). FDD ist eine spezielle Entwicklungsmethodik, die auf feingranularen Features basiert. Zwar gibt es begriffliche und methodische Überschneidungen zu APM, aber die Unterschiede sind wahrscheinlich größer als die Gemeinsamkeiten, sodass wir eine detaillierte Abgrenzung zum Ansatz von DeLuca für nicht relevant halten.

FDD

Auch zu dem Buch „Agile Project Management“ von Jim Highsmith [Highsmith-2004], das gelegentlich auch „APM“ abgekürzt wird, hat unser Buch im Übrigen keinen besonderen Bezug, außer dass das Thema identisch ist.

APM